UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/721,435	11/25/2003	Vampo Cosimo	FR920030008US1	7141

7590 06/22/2006

Jeffrey S. LaBaw
International Business Machines
11400 Burnet Rd.
Austin, TX 78758

EXAMINER	
NEWAY, SAMUEL G	

ART UNIT	PAPER NUMBER
2194	

DATE MAILED: 06/22/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	10/721,435	COSIMO ET AL.	
	Examiner	Art Unit	
	Samuel G. Neway	2194	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *25 November 2003*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-13* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-13* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *25 November 2003* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☒ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☒ All b)☐ Some * c)☐ None of:

1.☒ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1 – 13 are pending and are considered below.

Specification

2. The disclosure is objected to because of the following informalities: In paragraph 43, the last line reads "... switched to the sign "31"". It is believed this is a typographical error and is being read by the Examiner as "... switched to the sign "-""for the prosecution of the application below.

Appropriate correction is required.

Claim Rejections - 35 USC § 101

3. 35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

Claims 9, 11, and 13 rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter. Computer programs claimed as computer listings *per se*, i.e., the descriptions or expressions of the programs, are not physical "things." They are neither computer components nor statutory processes, as they are not "acts" being performed. Such claimed computer programs do not define any structural and functional interrelationships between the computer program and other claimed elements of a computer which permit the computer program's functionality to be realized. In contrast, a claimed computer-readable medium encoded with a computer

program is a computer element which defines structural and functional interrelationships between the computer program and the rest of the computer which permit the computer program's functionality to be realized, and is thus statutory.

Claim Rejections - 35 USC § 102

4. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

5. Claims 1, 6, and 9 – 13 are rejected under 35 U.S.C. 102(e) as being anticipated by Chenier (USPGPub 2004/0003383).

6. Claims 1, and 9 - 13: Chenier discloses a method of editing program code on a data processing system, the program code being suitable for subsequent processing, wherein the method includes the steps of:

defining at least one portion of the program code (paragraph 5),

selecting at least one defined portion (paragraph 5),

and automatically disabling the at least one selected portion, the at least one disabled portion being excluded from the subsequent processing (paragraph 5).

Claim 6: Chenier discloses the method according to claim 1, wherein the step of defining the at least one portion includes: enclosing each portion between a starting

comment and an ending comment ("...replace the command with comments", paragraph 33).

Claim Rejections - 35 USC § 103

7. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

8. Claims 2 – 5, and 7 – 8 are rejected under 35 U.S.C. 103(a) as being unpatentable over Chenier in view of Endejan (USPGPub 2002/0184611).

9. Claim 2: Chenier discloses the method according to claim 1, but does not disclose further including the steps of: selecting at least one previously disabled portion, and automatically re-enabling the at least one selected previously disabled portion. Endejan discloses a similar system with an editor displaying active and inactive pieces of code in separate displays. Endejan also discloses selecting at least one previously disabled portion, and automatically re-enabling the at least one selected previously disabled portion ("... from the inactive display format to the active display format...", paragraph 29). Therefore, it would have been obvious to one with ordinary skill in the art at the time the invention was made to automatically re-enable portions of code that had previously been disabled. One would have been motivated to select and re-enable some portions of code to better maintain the code by reinstating comments or debugging capabilities if for example some formally unforeseen or unsuspected bug was discovered.

Claim 3: Chenier and Endejan disclose the method according to claim 2, Chenier also discloses further including the step of:

assigning each defined portion to a category of a set including at least one category ("... identifies the code element and the comment element...", see Abstract),

the step of selecting the at least one defined portion (paragraph 5)

and the step of selecting the at least one previously disabled portion including selecting at least one category (paragraph 5).

Claim 4: Chenier and Endejan disclose the method according to claim 3, Chenier also discloses the set including at least one category for service instructions ("... code in a program for testing and debugging purposes." paragraph 3).

Claim 5: Chenier and Endejan disclose the method according to claim 2, Chenier also discloses the program code including a plurality of instructions, the step of automatically disabling the at least one selected portion including converting each corresponding instruction into a comment (paragraphs 5, 33), but he does not disclose the step of automatically re-enabling the at least one selected previously disabled portion including restoring each corresponding instruction. Endejan discloses selecting at least one previously disabled portion, and automatically re-enabling the at least one selected previously disabled portion ("... from the inactive display format to the active display format...", paragraph 29). Therefore, it would have been obvious to one with ordinary skill in the art at the time the invention was made to automatically re-enable portions of code that had previously been disabled. One would have been motivated to select and re-enable some portions of code to better maintain the code by reinstating

comments or debugging capabilities if for example some formally unforeseen or unsuspected bug was discovered.

Claim 7: Chenier and Endejan disclose the method according to claim 2, Chenier discloses further including the step of: condensing each disabled portion by at least partially hiding a corresponding visual representation (paragraph 5), but he does not disclose restoring the visual representation of each re-enabled portion. Endejan discloses selecting at least one previously disabled portion, and automatically re-enabling the at least one selected previously disabled portion ("... from the inactive display format to the active display format...", paragraph 29). Therefore, it would have been obvious to one with ordinary skill in the art at the time the invention was made to automatically re-enable portions of code that had previously been disabled. One would have been motivated to select and re-enable some portions of code to better maintain the code by reinstating comments or debugging capabilities if for example some formally unforeseen or unsuspected bug was discovered.

Claim 8: Chenier and Endejan disclose the method according to claim 7, Chenier also discloses further including the steps of: updating the program code by removing each condensed portion ("...information stripped out...", paragraph 5), and storing the updated program code ("...source code to be stored...", paragraph 33).

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Samuel G. Neway whose telephone number is 571-270-1058. The examiner can normally be reached on Mon - Thur 8:00AM - 5:00 PM EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, James Myhre can be reached on 571-270-1065. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

SGN
SGN
06/05/2006

James W. Myhre
Supervisory Patent Examiner

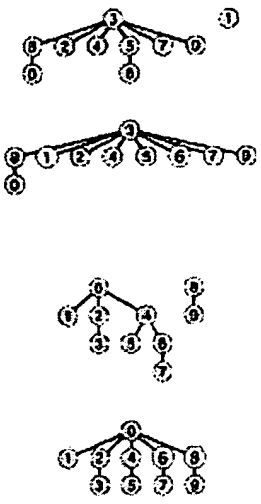

Figure 1.9
Path compression

We can make paths in the trees even shorter by simply making all the objects that we touch point to the root of the new tree for the union operation, as shown in these two examples. The example at the top shows the result corresponding to Figure 1.7. For short paths, path compression has no effect, but when we process the pair 1 6, we make 1, 5, and 6 all point to 3 and get a tree flatter than the one in Figure 1.7. The example at the bottom shows the result corresponding to Figure 1.8. Paths that are longer than one or two links can develop in the trees, but whenever we traverse them, we flatten them. Here, when we process the pair 6 8, we flatten the tree by making 4, 6, and 5 all point to 0.

tions to process M edges on N objects (see Exercise 1.9). This result is in stark contrast to our finding that quick find always (and quick union sometimes) uses *at least* $MN/2$ instructions. The conclusion is that, with weighted quick union, we can guarantee that we can solve huge practical problems in a reasonable amount of time (see Exercise 1.11). For the price of a few extra lines of code, we get a program that is literally millions of times faster than the simpler algorithms for the huge problems that we might encounter in practical applications.

It is evident from the diagrams that relatively few nodes are far from the root; indeed, empirical studies on huge problems tell us that the weighted quick-union algorithm of Program 1.3 typically can solve practical problems in *linear* time. That is, the cost of running the algorithm is within a constant factor of the cost of reading the input. We could hardly expect to find a more efficient algorithm.

We immediately come to the question of whether or not we can find an algorithm that has *guaranteed* linear performance. This question is an extremely difficult one that plagued researchers for many years (see Section 2.7). There are a number of easy ways to improve the weighted quick-union algorithm further. Ideally, we would like every node to link directly to the root of its tree, but we do not want to pay the price of changing a large number of links, as we did in the quick-union algorithm. We can approach the ideal simply by making all the nodes that we do examine link to the root. This step seems drastic at first blush, but it is easy to implement, and there is nothing sacrosanct about the structure of these trees: If we can modify them to make the algorithm more efficient, we should do so. We can easily implement this method, called *path compression*, by adding another pass through each path during the *union* operation, setting the id entry corresponding to each vertex encountered along the way to link to the root. The net result is to flatten the trees almost completely, approximating the ideal achieved by the quick-find algorithm, as illustrated in Figure 1.9. The analysis that establishes this fact is extremely complex, but the method is simple and effective. Figure 1.11 shows the result of path compression for a large example.

There are many other ways to implement path compression. For example, Program 1.4 is an implementation that compresses the paths by making each link skip to the next node in the path on the way up the tree, as depicted in Figure 1.10. This method is slightly easier to

then $f^\circ = foldrn\ step\ base$.

We will postpone the proof of Theorem 1 to Sect. 7, where in fact a more general result is proved. For now, let us see some of its applications.

4 Building a Tree from Its Depths

We will start with a formal specification of the problem of a building a tree given the depths of its tips. First of all, the familiar function *flatten*, which takes a tree and returns its tips in left-to-right order, can be written as a fold:

$$\begin{aligned} &flatten :: Tree\ A \to List^+\ A \\ &flatten = foldtree\ (+\!\!+)\ wrap \end{aligned}$$

Here $wrap\ x = [x]$ wraps an item into a singleton list.

A tree of integers is *well-formed* if one can assign to it a *level*, where the level of a tip is the number at the tip, and the level of a non-tip is defined only if its two subtrees have the same level, in which case it is one less than the levels. The partial function *level* can be defined by:

$$\begin{aligned} &level :: Tree\ Int \rightarrowtail Int \\ &level = foldtree\ up\ id \\ &\qquad \textbf{where}\ up\ a\ b = \textbf{if}\ a = b\ \textbf{then}\ a - 1 \end{aligned}$$

Note that the **if** clause in the definition of *up* has only one branch. Therefore, *level* is a partial function which only returns a value for a tree when its left and right subtrees have been assigned the same level.

We call a tree *well-formed* if it is in the domain of *level*. Our problem can thus be specified by

$$build = dom\ level \cdot flatten^\circ$$

We have generalised the problem a little, allowing the level number of the resulting tree to be other than zero.

The relation $flatten^\circ$ maps a list to an arbitrary tree that flattens to the list. For a given list, there will be many such trees. The coreflexive *dom level* acts as a filter picking those that are well-formed. Our specification is therefore an instance of the "generator - filter" paradigm that recurs frequently in functional programming.

Now we have got the problem specification, we are left with two problems: how to compute $flatten^\circ$, and how to fuse *dom level* into the computation.

4.1 Building a Tree with a Fold

Our aim is to apply the converse-of-a-function theorem to invert *flatten*. We need a pair of relations $one :: A \rightsquigarrow Tree\ A$ and $add :: A \to Tree\ A \rightsquigarrow Tree\ A$ that are jointly surjective and satisfy

$$\begin{aligned} flatten\ (one\ a) &= [a] \\ flatten\ (add\ a\ x) &= a : flatten\ x \end{aligned}$$

The action:

```
<xsl:copy-of select="$Contents"/>
```

copies the <ul> element and all its children to the result tree, producing a bulleted list in the final HTML page. In contrast, the action:

```
<xsl:value-of select="$Contents"/>
```

just copies the text `OneTwo` to the result tree, since the value of an element is defined by the concatenation in document order of all of its text node descendants.

Having a result tree fragment containing HTML nodes is different from having a single text element containing the "tags and text" for HTML, as we saw earlier in this chapter with our web store product blurb. In the web store case, we had a text chunk with tags in it that we needed to insert verbatim in the result tree so that the browser would ultimately interpret the angle brackets in the HTML text blurb as tags. In the <ul> result tree fragment case here, the processor is working with a tree of nodes and not just a text chunk containing tags. This explains why in the web store example our:

```
<xsl:value-of disable-output-escaping="yes" select="BLURB"/>
```

copies a single text node containing angle brackets and writes the angle brackets verbatim to the output, while:

```
<xsl:value-of select="$Contents"/>
```

flattens the result tree fragment into the string `OneTwo` by concatenating all of the text nodes in the tree fragment.

With our *TitledBox.xsl* stylesheet in place, we can begin creating our news portal. We start by recalling the tables that sit underneath our news portal. We organize our news stories into content categories, so we have a site_newscategory table.

```
CREATE TABLE site_newscategory(
  id   NUMBER,
  name VARCHAR2(80),
  CONSTRAINT site_newscatpk PRIMARY KEY (id)
);
```

Building Oracle XML Applications

Steve Muench

9/2000